WSS-10502/29
31711sh

VIRTUAL OS COMPUTING ENVIRONMENT

REFERENCE TO RELATED APPLICATION

This application claims priority from U.S. Provisional Patent Application Serial No. 60/427,339, filed November 18, 2002, the entire content of which is incorporated herein by reference.

FIELD OF THE INVENTION

This invention relates to computing environments and, in particular, to the creation of multiple, semi-independent virtual operating system (OS) environments within a single OS.

BACKGROUND OF THE INVENTION

An Operating System (OS) is the high-level software that allows users to interact with a computing machine and run programs (i.e., *applications*). To enhance the versatility of computers, virtual operating systems (VOS) and programs that implement virtual machines (VM) have been developed. Using such systems, applications "expecting" a particular OS may run on a machine executing a different OS, or commands generated by an application may cause a real computer to act like the imaginary machine. Theoretically, a VOS may allow the same program to work on virtually any machine running the Virtual OS, thereby supporting "instant portability" for new machines.

The concept of a virtual machine has been popularized by Pascal compilers which produce intermediate "p-code" and, more recently, by Java language from Sun Microsystems. Whereas most compilers produce object code for one family of CPU, Java compilers produce object code (called J-code) for machines that may or may not exist. For each physical target processor, a Java interpreter, or virtual machine, "executes" the J-code. This allows the same object code to run on any CPU for which a Java interpreter exists.

Other examples include Limbo, a programming language developed at Lucent Technologies, produces object code for an imaginary CPU, and Perl, which creates an intermediate program representation and executes this intermediate representation instead of creating native executable code.

The major standard operating systems, namely Windows, Macintosh and Unix, make very different "calls" to the OS. These calls are critical to writing sophisticated applications, which limits portability. Java solves this problem by providing a set of library functions that communicate with an imaginary OS and imaginary GUI (graphical user interface). In a sense, just like the JVM presents a virtual physical machine, the Java libraries present a virtual OS/GUI. Every Java implementation provides libraries implementing this virtual OS/GUI. Java programs that use these libraries to provide needed OS and GUI functionality port fairly easily.

Virtual operating systems include the Amiga OS, which is based on the Taos Elate OS, and Inferno developed by Lucent Technologies. Inferno, a virtual operating system running on top of the Linux operating system, is targeted to creating and supporting distributed services used in a variety of network environments, including advanced telephones, hand-held devices, TV/cable/satellite set-top boxes, Internet computers and conventional computing systems. Applications use various resources internal to the system, such as a consistent virtual machine that runs the application programs, together with library modules that perform services as simple as string manipulation through more sophisticated graphics services for dealing with text, pictures, higher-level toolkits, and video. Applications exist in an external environment containing resources such as data files that can be read and manipulated, together with objects that are named and manipulated like files but are more active.

SUMMARY OF THE INVENTION

This invention is directed to the creation of multiple, semi-independent virtual OS environments within a single operating system (OS). A change made in one environment does not affect the main OS or any other environment. In this way each virtual OS

environment appears to be an independent OS for the applications running within it. The file system and registry information for each environment is independent of the base OS and other environments. Each of the environments can contain a group of installed applications that will run independently of each other. Although the invention is described in terms of a Windows® environment, the approach is applicable to other operating systems through appropriate modification.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGURE 1 is a simplified diagram showing the creation of multiple OS environments under a single OS;

FIGURE 2 is a illustrates the creation of independent locations for file systems and registry within the base OS file system and registry;

FIGURE 3 shows how, according to the invention, injected DLL functions can execute code both before and after the OS API function;

FIGURE 4 shows how any applications that are run from under the virtual OS environment file system are always redirected into that virtual OS environment;

FIGURE 5 depicts when application is being redirected into a virtual OS environment, it makes an API call that tries to access the file system or registry, causing the API to be redirected to a function in the injected DLL;

FIGURE 6A is a diagram that shows how getting and setting of the current directory are redirected into the virtual OS environment.

FIGURE 6B shows how injected DLL functions maintain lists of these handles and the pathnames that they reference. These lists are then used in subsequent calls to OS API functions to modify the parameters properly; and

FIGURE 7 is a diagram that depicts how OS API calls can be redirected by the injected DLL function modifying a parameter sent to or returned from the OS API.

DETAILED DESCRIPTION OF THE INVENTION

This invention is directed to the creation of multiple, semi-independent virtual OS environments within a single operating system (OS). A change made in one environment does not affect the main OS or any other environment. In this way each virtual OS environment appears to be an independent OS for the applications running within it. The file system and registry information for each environment is independent of the base OS and other environments. Each of the environments can contain a group of installed applications that will run independently of each other. Although the invention is described in terms of a Windows® environment, the approach is applicable to other operating systems through appropriate modification.

The applications running within the virtual OS environments still share the base OS attributes such as networking information, user login rights, services, hardware information, and the Windows clipboard. In addition, all of the applications run on a single OS desktop. The end user does not need to be aware that the applications are being run from different virtual OS environments. While the applications share the base OS attributes, the changes made to configuration information is made into the virtual OS environment and not into the base OS.

The virtual OS environment is achieved by creating independent locations for file systems and registry within the base OS file system and registry. When applications attempt to access the file system or registry, the attempt is redirected to the virtual OS environment file system or registry instead of the base OS location.

To create the virtual OS environments, the OS APIs that access the file system and registry directly and indirectly must be changed to redirect these accesses into the virtual OS environments file system and registry. This is done by injecting a DLL into every application that is executed. This DLL first must determine whether the current application should be run under one of the possible virtual OS environments or the base OS. If the current application is to run under the base OS the injected DLL does nothing and the application runs normally. If the current application needs to be redirected into a virtual OS environment, the injected DLL scans the applications function import table and redirects all file system and registry API calls to instead call functions within the

injected DLL itself. This is done for the application functions and all of the DLLs that are loaded by the application. In addition, Windows COM calls that access the file system or registry are also redirected by the injected DLL.

When the application (or one of the applications DLLs) attempts to call an OS API (directly or indirectly) that will access the file system or registry, the injected DLL's function is called instead. The injected DLL's function then examines the parameters that are passed to the API and modifies them to direct them into the virtual OS environment location instead. The injected DLL's function then calls the real OS API with the modified parameters to perform the required function. Lastly, the injected DLL's function returns to the calling function, returning any information from the OS API function. This returned information is also modified to convert the file system or registry location information back from a virtual OS environment location. Using this method, the injected DLL functions can execute code both before and after the OS API function.

As mentioned earlier, not all applications are run from virtual OS environments and different applications may be run from different virtual OS environments. When the injected DLL is first loaded it must determine which of the virtual OS environments (if any) to redirect to. This must be done for the current application as well as any spawned or child processes created by the current application. This is determined based on the location of the current application EXE in the base OS file system. The base OS contains a list of directories where applications should be redirected to specific virtual OS environments. Any application EXE that is executed in this directory or optionally a child of this directory will run under a specific virtual OS environment. This is can be set to a CD/DVD drive in the base OS file system and the application run from this location is often a application installation program. Note that this directory is added to the virtual OS environment. File system accesses to this directory (or children) will not be redirected but will be performed in the directory itself. This directory (or directories) are shared by the base OS and the virtual OS environment. This allows the installation program or application to access support files that are then installed into the virtual OS environment.

In addition, any applications that are run from under the virtual OS environment file system (ex: C:\VirtualEnvs\Env1) are always redirected into that virtual OS environment. Since the temporary directory is also redirected into the virtual OS environment, if an installation program extracts additional installation programs from itself into a temporary directory and then executes them, they will be redirected into the same virtual OS environment. Since the installation program can only make changes to the virtual OS environment and the shared directory, it can only place files into a location that is under the virtual OS environment itself. It cannot change any file system or registry information in the base OS file system or registry except in shared directories. This method guarantees that the proper applications are always running in the proper virtual OS environment or base OS and that the applications can only modify specific areas of the base OS file system and registry.

Because all of the file system and registry calls are redirected, a copy of the base OS file system and registry must be created in the virtual OS environment file system and registry before any applications may be executed in the virtual OS environment. Once this is done, any built-in OS applications that are run by applications within the virtual OS environment (such as Notepad, WordPad, or Internet Explorer) will be executed from the virtual OS environment copy of that application. This will allow multiple versions of built-in OS applications to be supported at the same time in different virtual OS environments.

When an application that is being redirected into a virtual OS environment makes an API call that tries to access the file system or registry, the API is redirected to a function in the injected DLL. This is done without any modifications required to the application or support DLLs.

This redirection is done for all OS API calls that access the file system. There are file system calls that use a relative path. These calls are relative to a current directory. The getting and setting of the current directory are redirected into the virtual OS environment. The application views that it is accessing/modifying the base OS file

system or registry when in fact it is accessing/modifying the virtual OS environment file system and registry.

The redirection of registry information is done the same way as file system information. All calls that access the registry are redirected into the injected DLL functions where they modify the parameters to redirect that registry access to a section of the registry that is private to the virtual OS environment. A number of registry and file system APIs are passed handles instead of a specific pathname. The injected DLL functions maintain lists of these handles and the pathnames that they reference. These lists are then used in subsequent calls to OS API functions to modify the parameters properly.

Since all of the virtual OS environment information is isolated inside of the base OS file system and registry, it becomes easy to save, store, and load entire virtual OS environments. It is therefore convenient to create a virtual OS environment that contains a set of applications configured in a specific way (or a "clean" virtual OS environment that contains no applications yet) and store it in a separate or central location to be used by multiple computers or the same computer at different times.

Not all OS API calls can be redirected by the injected DLL function modifying a parameter sent to or returned from the OS API. For example when creating a shortcut using the Program Manager DDE calls, the shortcut is created in the base OS file system (after backing up any shortcut in that same location) and then moved into the virtual OS environment file system. Afterward, any shortcut that was in the base OS file system is then restored. This occurs within the injected DLL function that is called from an application in the virtual OS environment.

Another special case occurs when a 16-bit application is to be executed within the virtual OS environment. The injected DLL does not function with 16-bit applications so this is normally not supported. In cases where the 16-bit application is known to only extract files from itself and then run a 32-bit application an exception is made. The 16-bit application is executed normally and allowed to place the extracted files into a temporary location in the base OS file system. This temporary location is then added as a shared

directory with the virtual OS environment and the 32-bit application that was extracted is then executed in the virtual OS environment.

Since the virtual OS environment file system files are not being used by the base OS itself, they can be replaced more easily than the actual base OS files that are normally in use (files that are in-use cannot normally be replaced except during a reboot). This also allows a reboot after an installation program to be handled without a reboot being required on the computer itself.

When the OS reboot API is called by a program that is running under a virtual OS environment, the injected DLL function performs the operations normally handled by the reboot without allowing the OS API to perform a system reboot. All of the applications in the virtual system are shutdown. The list of files that are queued to be replaced during a reboot are then installed. The list of applications that are normally run during a system startup are then executed in order. This allows application installation programs that contain require one or more reboots function properly without a reboot actually being performed.

I claim: